

## گزارش فاز ۲ پروژه برنامه نویسی چند هسته ای

محمد حیدری راد

۹۹۳۱۰۱۷

رضوان آفری

۹۸۲۷۰۰۳

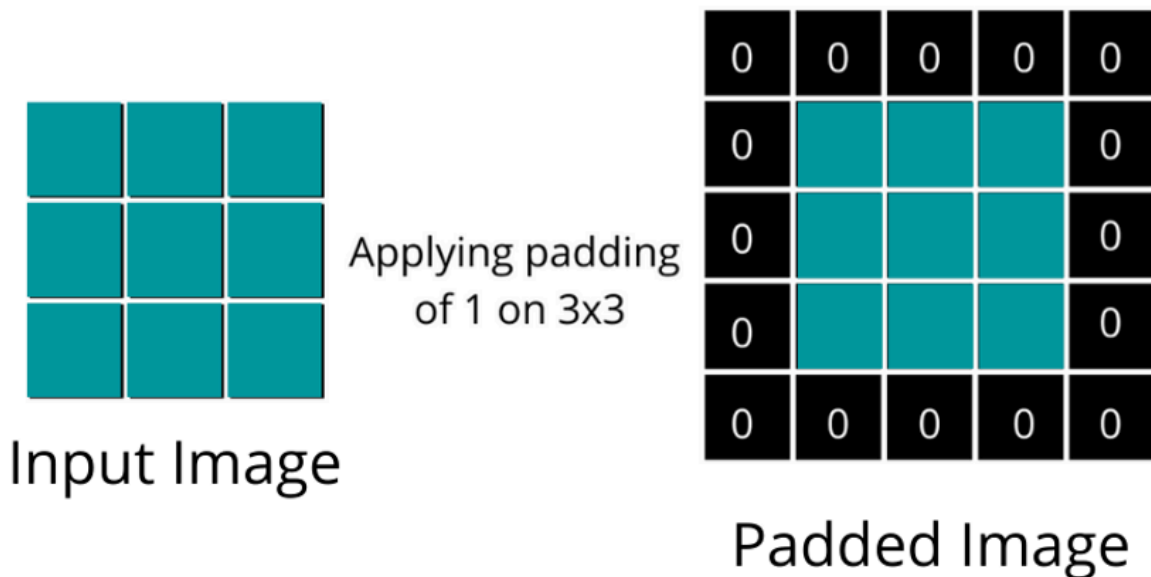
### پیاده سازی شبکه عصبی vgg-16 به وسیله CUDA.

#### توضیحات پیاده سازی

ورودی این مدل شبکه عصبی، یک تصویر با ابعاد  $3 \times 224 \times 224$  است و خروجی یک بردار 1000 تایی. برای پیاده سازی این مدل شبکه عصبی، نیاز به پیاده سازی 4 تا kernel داریم که شرح داده میشوند.

#### ۱- پیاده سازی padding:

خروجی این کار همانطور که در شکل زیر مشاهده میشود، به صورتی است که به اندازه pad\_w تا به دیواره های عمودی هر channel در تصویر خانه های 0 اضافه میشود و به اندازه pad\_h تا به دیواره های افقی. در شکل زیر یک نمونه از padding بر یک تصویر دارای یک channel و ابعاد  $3 \times 3$  را میتوان مشاهده کرد که pad\_h و pad\_w هر دو برابر 1 هستند.



جزئیات پیاده سازی:

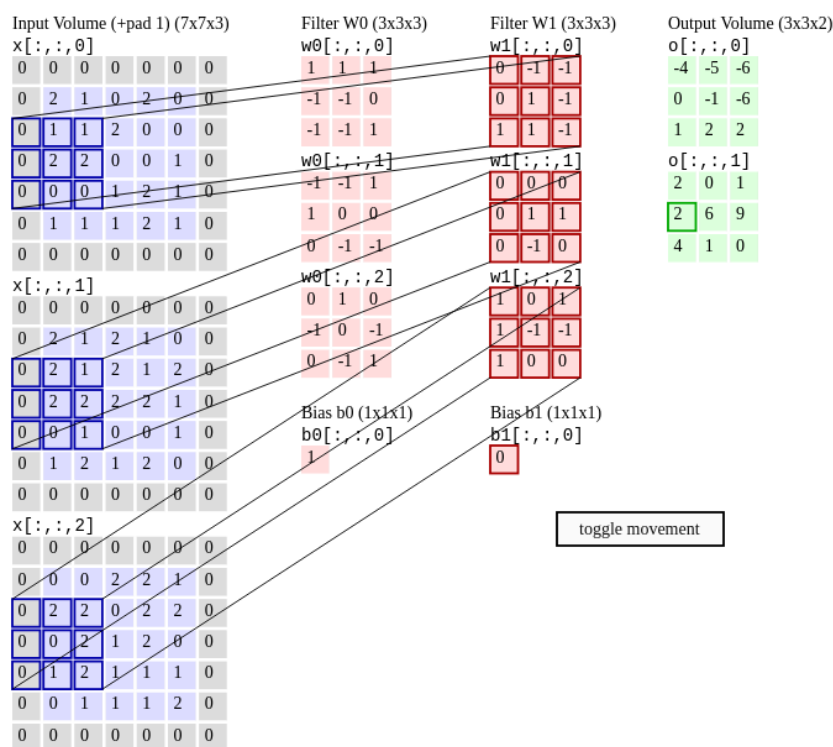
کد **kernel** پیاده سازی شده به این صورت عمل میکند که هر نخ، مسئولیت پر کردن یک خانه از تصویر را دارد، اگر موقعیت نخ داخل تصویر خروجی نبود کاری نمیکند، اما اگر داخل تصویر خروجی بود، حال چک میکند که آیا داخل **padding** هست یا خیر، اگر بود، خانه مورد نظر خود را برابر 0 قرار میدهد، اگر داخل **padding** نبود خانه متناظر خود را از تصویر ورودی برداشته و در خانه مورد نظر خود در تصویر خروجی قرار میدهد. کد **kernel** برای **padding** در تابع **pad\_kernel** پیاده سازی شده.

همچنین برای استفاده از **padding** با استفاده از این **kernel**، تابعی به نام **pad\_with\_cuda** پیاده سازی شده که کارهای محاسبه اندازه تصویر خروجی، مقدار دهی به **gridDim** و **blockDim**، انجام **kernel lunch** و برگرداندن تصویر خروجی را انجام میدهد.

## ۲- پیاده سازی conv\_2D:

در لایه **CNN**، یک تصویر به ابعاد  $input\ channels * height * width$  میگیریم و یک تصویر با ابعاد  $output\ channels * (height - kernel\ height + 1) * (width - kernel\ width + 1)$  خروجی میدهیم.

انجام این کار در تصویر زیر قابل مشاهده است:



یک **kernel** داریم که ابعاد

$output\ channels * input\ channels * kernel\ width * kernel\ height$  دارد.

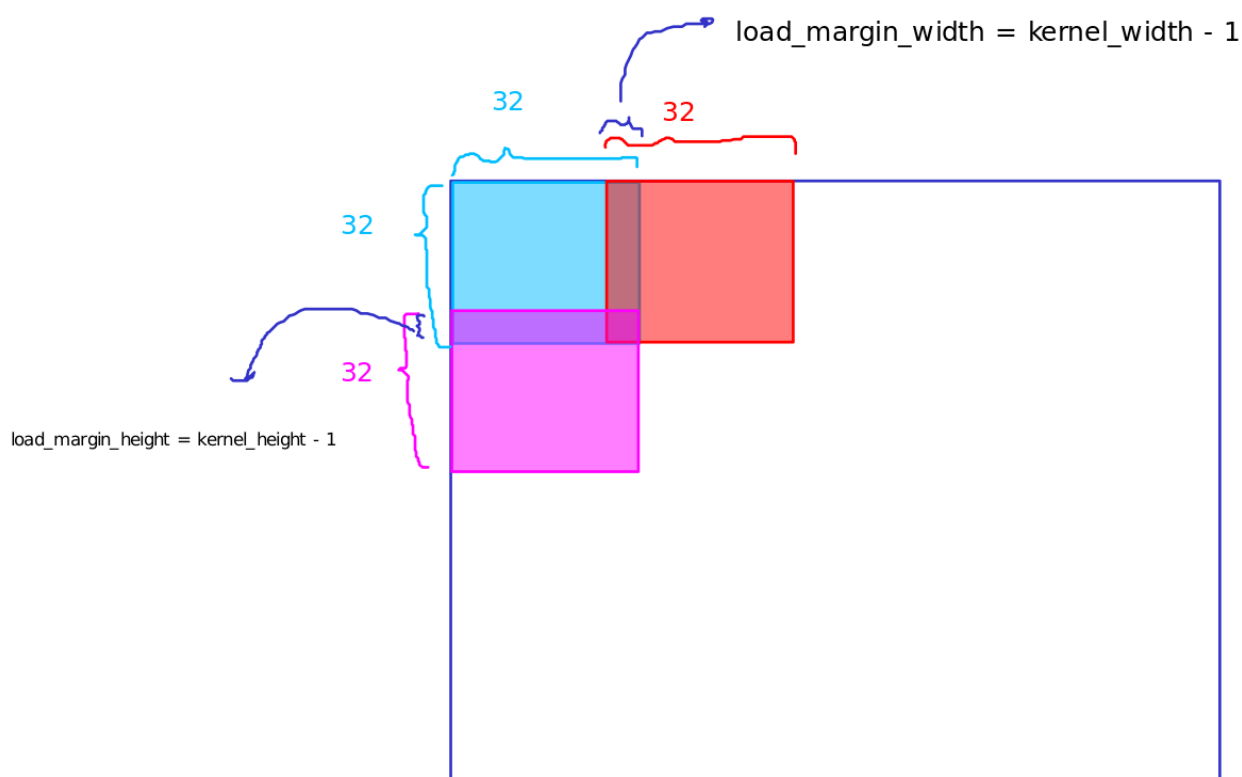
حال به ازای هر **output channel** باید آن بخش از **kernel** را در کل ماتریس تصویر به صورت **convolution** ای ضرب کنیم و نتیجه را با **bias** مورد نظر آن **output channel** جمع کنیم، سپس تابع **relu** را روی آن اجرا کنیم (بیشینه 0 و عدد به دست آمده) و در مکان مورد نظر در تصویر خروجی قرار دهیم.

جزئیات پیاده سازی:

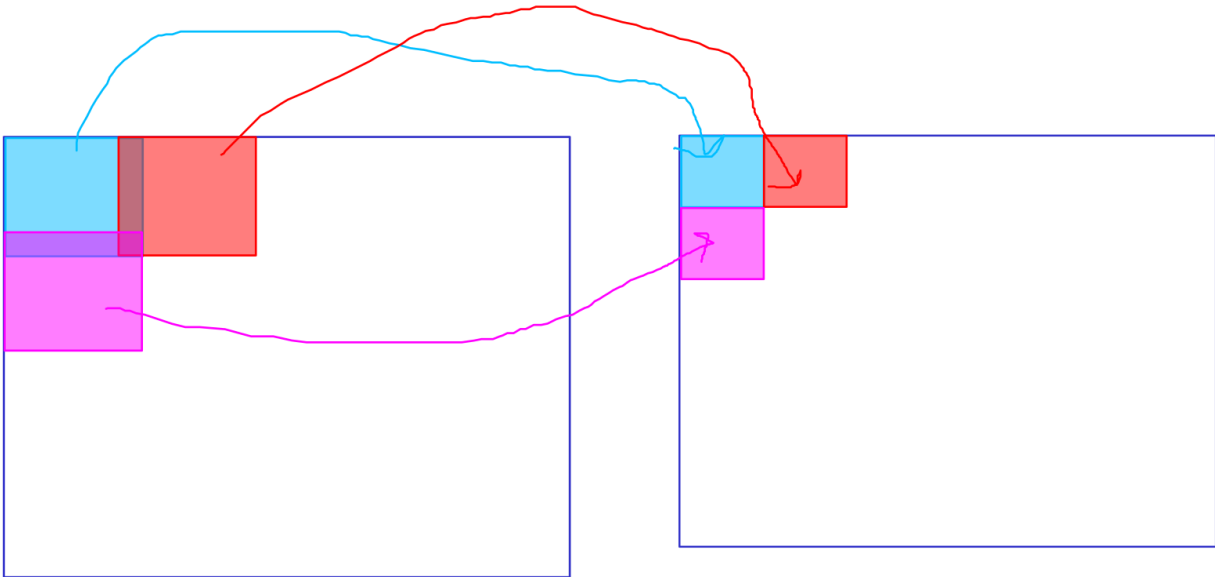
برای انجام این کار، هر block یک تکه  $32 \times 32$  از تصویر را در خود load میکند، اما این تکه ها با هم overlap دارند، به صورت افقی به اندازه  $(pad\_w - 1)$  و به صورت عمودی به اندازه  $(pad\_h - 1)$ .

حال در هر block، هر نخ مسئولیت محاسبه یک pixel در خروجی را به عهده میگیرد (نخ هایی که در load margin هستند محاسبه ای دیگر انجام نمیدهند و کارشان صرفا load کردن خانه های overlap شده با block های کناری بوده).

هر نخ به ازای هر input channel، بخش مورد نظر خود را متناظرا در kernel ضرب میکند و نتیجه را با bias مورد نظر جمع میکند و تابع relu را روی آن اجرا کرده و در مکان مورد نظر خود در تصویر خروجی قرار میدهد. تابع kernel پیاده سازی شده برای conv\_2D در فایل با نام conv\_2D\_kernel پیاده سازی شده. نحوه عملکرد به صورت load\_margin در شکل زیر قابل مشاهده است:



در نهایت هر block بخش محاسبه خود را در تصویر خروجی مینویسد.

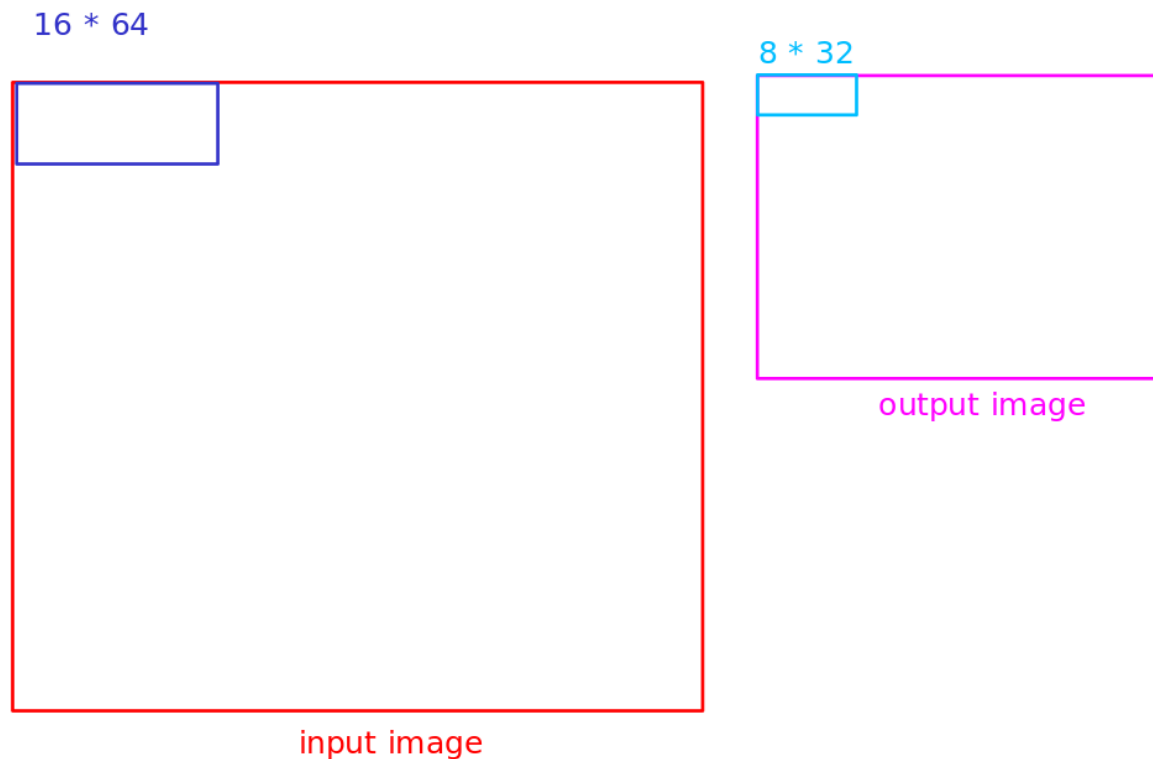


### ۳- پیاده سازی max\_pooling:

به این صورت عمل میکنیم که هر block از نخ ها، به صورت یک block با ابعاد  $64 * 16$  است (در بعد x دارای 64 نخ و در بعد y دارای 16 نخ است). و هر نخ در block، مسئول load کردن خانه مورد نظر خود است (بسته به اینکه در کدام channel قرار دارد و اینکه سطر و ستونش چیست).

حال که این 1024 نخ خانه مورد نظر خود را از حافظه در shared memory قرار دادند. نخ های 0 الی 31 در ردیف های 0 تا 8 به سراغ محاسبه max pooling درایه های مورد نظر خود میروند. علت این نوع تقسیم بندی  $64 * 16$  ای به جای تقسیم بندی  $32 * 32$  ای، این است که چون اندازه ماتریس قرار است نصف شود، با تقسیم بندی  $32 * 32$  در هر warp صرفا 16 تا نخ در حال محاسبه max pooling هستند و 16 نخ دیگر بیکار هستند، و در این 32 ردیف نیز 16 تا warp فعال هستند، یعنی 16 تا warp نیمه فعال داریم. اما با تقسیم بندی  $64 * 16$ ، warp های نیمه سمت راست کاملا غیر فعال هستند، همینطور warp های نیمه پایین نیز کاملا غیر فعال هستند، (منظور از غیر فعال، این است که صرفا تا مرحله load کردن در shared memory فعالیت دارند و بعد از آن کاری انجام نمیدهند). پس در این شرایط 8 تا warp تماما فعال داریم به جای اینکه 16 تا warp نیمه فعال باشیم و این برای ما مطلوب است.

در شکل زیر می توان max pooling توصیف شده را مشاهده کرد:



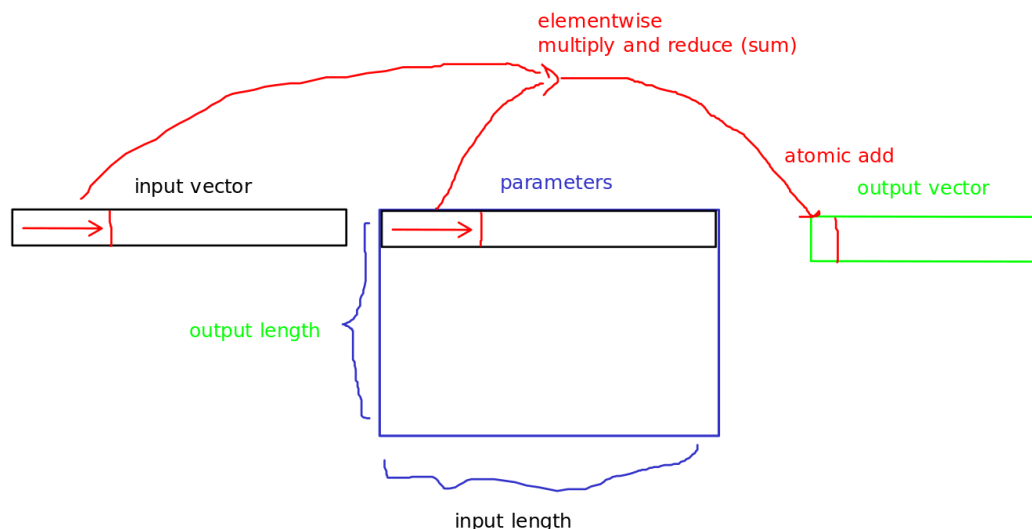
۴- پیاده سازی FC\_relu: برای این بخش، از تعداد بیشتری kernel استفاده شده که هر کدام را جداگانه توضیح می‌دهیم:

**کرل zero:** این کرل یک بردار (یا ماتریس، یا تصویر یا...) را به همراه طول آن گرفته و تمامی درایه های آن را برابر 0 قرار می‌دهد.

**کرل relu:** این کرل یک بردار را به همراه طول آن گرفته و روی تمامی درایه های آن تابع relu را اجرا میکند (اگر درایه منفی است، آن را برابر 0 قرار می‌دهد).

**کرل add\_with\_vector\_1:** این کرل دو بردار را به همراه طول آن ها (که یکسان هستند) گرفته و بردار اولی را برابر حاصل جمع متناظر درایه به درایه دو بردار قرار می‌دهد.

**کرل FC:** این کرل، یک بردار ورودی به طول  $input\ length$ ، یک بردار خروجی به طول  $output\ length$  و یک ماتریس  $parameters$  به ابعاد  $output\ length * input\ length$  را گرفته، و به صورت زیر بردار خروجی را پر میکند:



مثالی از بلوک 16 نخی و warpsize=2

برای پیاده سازی FC\_relu، ابتدا بردار خروجی به وسیله zero\_all تماماً صفر میشود، سپس به وسیله FC مقادیری که توضیح داده شد در بردار خروجی محاسبه میشوند، سپس به وسیله کرنل add\_with\_vector\_1 مقادیر با bias مورد نظر جمع زده میشوند و سپس به وسیله کرنل relu تابع فعال ساز relu روی درایه های بردار خروجی انجام میشود.

زمان اجرا

زمان اجرای کد اصلی:

```
mohammad at egovivo in /media/mohammad/RADHDD/Documents/doroos
❯ ./vgg.out
CONV 224x224x64 11213 ms
CONV 224x224x64 102 ms
POOLMAX 112x112x64 31386 ms
CONV 112x112x128 106 ms
CONV 112x112x128 84 ms
POOLMAX 56x56x128 135 ms
CONV 56x56x256 105 ms
CONV 56x56x256 82 ms
POOLMAX 28x28x256 1163 ms
CONV 28x28x512 103 ms
CONV 28x28x512 103 ms
POOLMAX 14x14x512 982 ms
CONV 14x14x1024 94 ms
CONV 14x14x1024 104 ms
POOLMAX 7x7x1024 703 ms
FC 4096 14507 ms
FC 4096 408 ms
FC 1000 18752 ms
total execution: 9.870460 seconds
```

زمان اجرای کد پیاده سازی شده با cuda:

```

mohammad at egovivo in /media/mohammad/RADHDD/Documents/doroos
O ./code.out
move image to global memory: 369.605000 MS
CONV: 224 * 224 * 64: 3.902000 MS
CONV: 224 * 224 * 64: 65.387000 MS
POOLMAX: 112 * 112 * 64: 0.565000 MS
CONV: 112 * 112 * 128: 34.016000 MS
CONV: 112 * 112 * 128: 64.555000 MS
POOLMAX: 56 * 56 * 128: 0.430000 MS
CONV: 56 * 56 * 256: 32.975000 MS
CONV: 56 * 56 * 256: 65.556000 MS
POOLMAX: 28 * 28 * 256: 0.248000 MS
CONV: 28 * 28 * 512: 33.794000 MS
CONV: 28 * 28 * 512: 54.279000 MS
POOLMAX: 14 * 14 * 512: 0.210000 MS
CONV: 14 * 14 * 1024: 72.884000 MS
CONV: 14 * 14 * 1024: 144.406000 MS
POOLMAX: 7 * 7 * 1024: 0.093000 MS
FC: 893.679000 MS
FC: 74.933000 MS
FC: 34.936000 MS
copied to cpu memory, total GPU time: : 1946.627000 MS
total execution: 1.947096 seconds

```

همانطور که مشاهده میشود، زمان اجرای کد پیاده سازی شده حدود 2 ثانیه و زمان اجرای کد cudnn برابر 10 ثانیه است.

پس 5 برابر speedup داشته ایم.

انجام profiling

بخش summary

میزان throughput حافظه، throughput محاسباتی، تعداد register های مصرفی، ابعاد grid و اندازه block به ازای هر kernel صدا زده شده در شکل زیر قابل مشاهده است.

بعضی از kernel ها (مانند add\_with\_vector\_1، zero) که کار محاسباتی زیادی ندارند و اندازه ورودی شان کم است در موارد بالا بسیار مصرف کمی دارند و زمان اجرای بسیار پایینی نیز دارند.



Page: Summary Result: 1 - 548 - conv2D_kernel Add Baseline Apply Rules Occupancy Calculator Source Comparison										
Current 548 - conv2D_kernel (8, 8, 64)(32, 32, 1) 3.29 usecond 4,499,579 30 0 - NVIDIA GeForce GTX 1650 Ti 1.37 cycle/usecond 7.5 [25023] phase_2_double.out										
This table shows all results in the report. Use the column headers to sort the results in this report. Double-click a result to see detailed metrics.										
ID	Estimated Speedup	Function Name	Demangled Name	Duration	Runtime Improvement (4.32651e+08)	Compute Throughput	Memory Throughput	# Registers	Grid Size	Block Size
0	50.69	pad_kernel	pad_kernel(double...	0.02	0.01	23.71	62.90	32	8, 8, 1	32, 32, 1
1	72.33	conv2D_kernel	conv2D_kernel(dou...	3.29	2.38	73.92	23.30	30	8, 8, 1	32, 32, 1
2	55.48	pad_kernel	pad_kernel(double...	0.34	0.19	25.35	80.32	32	8, 8, 1	32, 32, 1
3	75.11	conv2D_kernel	conv2D_kernel(dou...	62.95	47.28	77.43	24.28	30	8, 8, 1	32, 32, 1
4	53.99	max_pool_2D_kernel	max_pool_2D_kern...	0.31	0.17	21.05	54.19	16	4, 14, 1	64, 16, 1
5	53.42	pad_kernel	pad_kernel(double...	0.69	0.05	25.29	77.87	32	4, 4, 1	32, 32, 1
6	75.00	conv2D_kernel	conv2D_kernel(dou...	31.45	23.59	77.36	24.24	30	4, 4, 1	32, 32, 1
7	54.37	pad_kernel	pad_kernel(double...	0.17	0.09	25.34	79.61	32	4, 4, 1	32, 32, 1
8	75.06	conv2D_kernel	conv2D_kernel(dou...	62.74	47.09	77.48	24.27	30	4, 4, 1	32, 32, 1
9	55.02	max_pool_2D_kernel	max_pool_2D_kern...	0.16	0.09	28.05	53.29	16	2, 7, 1	64, 16, 1
10	50.46	pad_kernel	pad_kernel(double...	0.04	0.02	25.06	76.76	32	2, 2, 1	32, 32, 1
11	75.50	conv2D_kernel	conv2D_kernel(dou...	31.27	23.62	77.53	24.30	30	2, 2, 2	32, 32, 1
12	50.99	pad_kernel	pad_kernel(double...	0.08	0.04	26.40	81.66	32	2, 2, 2	32, 32, 1
13	75.23	conv2D_kernel	conv2D_kernel(dou...	62.62	47.11	77.44	24.28	30	2, 2, 2	32, 32, 1
14	50.64	max_pool_2D_kernel	max_pool_2D_kern...	0.08	0.04	26.69	40.70	16	2, 4, 2	64, 16, 1
15	49.90	pad_kernel	pad_kernel(double...	0.02	0.01	25.55	70.15	32	1, 1, 2	32, 32, 1
16	75.20	conv2D_kernel	conv2D_kernel(dou...	31.28	23.62	77.68	24.33	30	1, 1, 5	32, 32, 1
17	51.20	pad_kernel	pad_kernel(double...	0.04	0.02	26.13	70.86	32	1, 1, 5	32, 32, 1
18	75.30	conv2D_kernel	conv2D_kernel(dou...	62.39	46.97	77.72	24.36	30	1, 1, 5	32, 32, 1
19	57.91	max_pool_2D_kernel	max_pool_2D_kern...	0.08	0.04	21.39	26.52	16	1, 2, 5	64, 16, 1
20	47.16	pad_kernel	pad_kernel(double...	0.03	0.02	23.43	23.91	32	1, 1, 5	32, 32, 1
21	64.48	conv2D_kernel	conv2D_kernel(dou...	72.99	47.07	66.55	25.09	30	1, 1, 10	32, 32, 1
22	48.11	pad_kernel	pad_kernel(double...	0.06	0.03	23.62	28.03	32	1, 1, 10	32, 32, 1
23	64.51	conv2D_kernel	conv2D_kernel(dou...	145.87	94.10	66.58	25.10	30	1, 1, 10	32, 32, 1
24	58.19	max_pool_2D_kernel	max_pool_2D_kern...	0.08	0.04	22.03	19.93	16	1, 1, 10	64, 16, 1
25	75.00	zero_all	zero_all(double * int)	0.00	0.00	2.30	7.38	16	4, 1, 1	1024, 1, 1
26	33.90	FC_kernel_improved	FC_kernel_improve...	77.74	26.35	66.58	26.53	16	49, 4096, 1	1024, 1, 1
27	75.00	add_to_vector_1_ke...	add_to_vector_1_ke...	0.00	0.00	3.81	14.07	16	4, 1, 1	1024, 1, 1
28	75.00	RELU_kernel	RELU_kernel(doubl...	0.00	0.00	3.98	8.95	16	4, 1, 1	1024, 1, 1
29	75.00	zero_all	zero_all(double * int)	0.00	0.00	2.31	7.32	16	4, 1, 1	1024, 1, 1
30	34.20	FC_kernel_improved	FC_kernel_improve...	6.31	2.16	67.12	26.74	16	4, 4096, 1	1024, 1, 1
31	75.00	add_to_vector_1_ke...	add_to_vector_1_ke...	0.00	0.00	3.65	13.37	16	4, 1, 1	1024, 1, 1
32	75.00	RELU_kernel	RELU_kernel(doubl...	0.00	0.00	4.01	9.09	16	4, 1, 1	1024, 1, 1
33	93.78	zero_all	zero_all(double * int)	0.00	0.00	0.57	2.23	16	1, 1, 1	1024, 1, 1
34	34.18	FC_kernel_improved	FC_kernel_improve...	1.58	0.54	66.95	26.67	16	4, 1000, 1	1024, 1, 1
35	93.78	add_to_vector_1_ke...	add_to_vector_1_ke...	0.00	0.00	0.99	3.52	16	1, 1, 1	1024, 1, 1
36	93.78	RELU_kernel	RELU_kernel(doubl...	0.00	0.00	1.04	2.64	16	1, 1, 1	1024, 1, 1

مشاهده میشود که طولانی ترین زمان های اجرا را conv\_2D ها دارند، مخصوصا conv2D آخر که ابعاد تصویر به 1024 \* 14 \* 14 میرسد و هر نخ باید به اندازه 1024 \* 3 \* 3 تا ضرب انجام دهد تا خانه مورد نظر خود در تصویر خروجی را محاسبه کند.

## بخش details

به ازای kernel های متفاوتی که launch شده، سوال های اشاره شده پاسخ داده میشوند.

سوال ها به ترتیب از (الف) الی (د) پاسخ داده میشوند.

از هر نوع kernel، صرفا یکی بررسی میشود.

کرنل 0:

کد: pad\_kernel

ابعاد ورودی: 3 \* 224 \* 224

ابعاد خروجی: 3 \* 226 \* 226

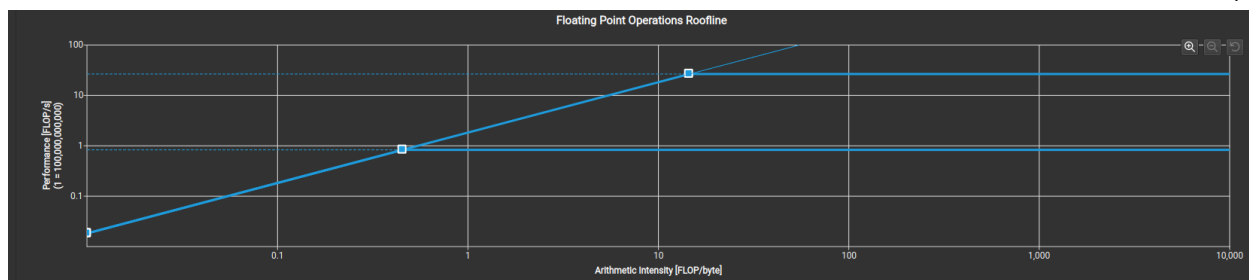
(الف)

Page: Details Result: 0 - 539 - pad_kernel Add Baseline Apply Rules Occupancy Calculator Source Comparison Copy as Image										
Current 539 - pad_kernel (8, 8, 3)(32, 32, 1) 17.15 usecond 22,312 32 0 - NVIDIA GeForce GTX 1650 Ti 1.30 cycle/usecond 7.5 [25023] phase_2_double.out										
GPU Speed Of Light Throughput GPU Throughput Rooflines										
High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.										
Compute (SM) Throughput [%]				Duration [usecond]				17.15		
Memory Throughput [%]				Elapsed Cycles [cycle]				22,312		
L1/TEX Cache Throughput [%]				SM Active Cycles [cycle]				19,469.88		
L2 Cache Throughput [%]				SM Frequency [cycle/usecond]				1.30		
DRAM Throughput [%]				DRAM Frequency [cycle/usecond]				5.72		

به علت اینکه این kernel هیچ کار محاسباتی ای انجام نمیدهد و صرفا با گرفتن تصویر ورودی، یک padding رویش انجام میدهد، compute throughput پایینی دارد.

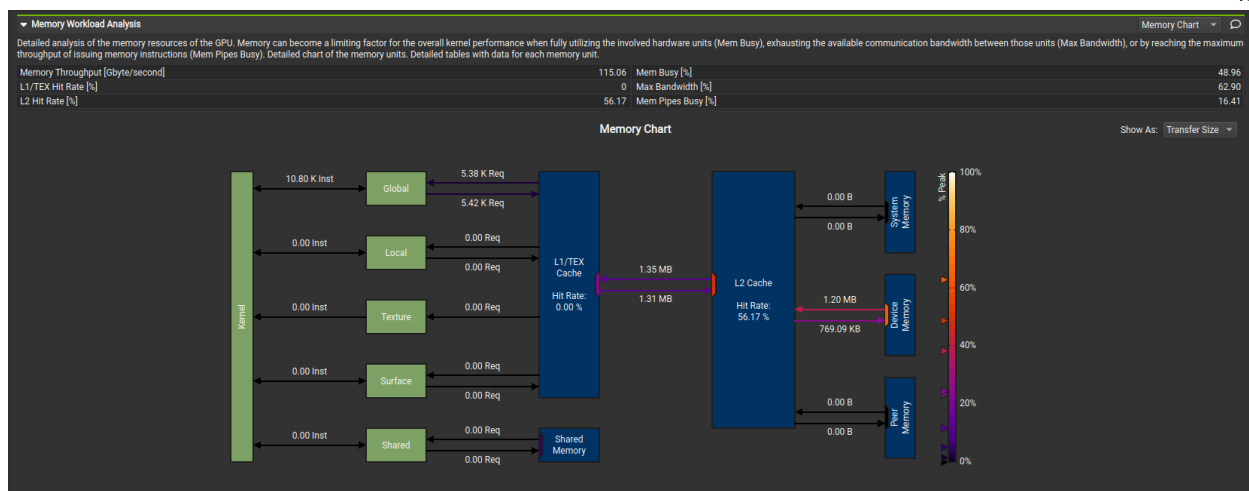
به همین علت میتوان "چشم بسته" گفت که این کرنل Memory bound است و compute throughput برابر 23.71% و Memory throughput برابر 62.90% ای همین امر را نشان میدهد.

(ب)



همانطور که مشاهده میشود، از آنجا که این kernel هیچ محاسبه floating point ندارد، در roofline model نیز نقطه ای برای آن نشان داده نمیشود، چرا که شدت حسابی اش برابر 0 است. چون ذاتا این kernel کار محاسباتی ای انجام نمیدهد، خیر جای بهبود برای افزایش شدت حسابی ندارد، مگر اینکه با kernel های دیگر ترکیب شود که کد را پیچیده میکند.

(ج)



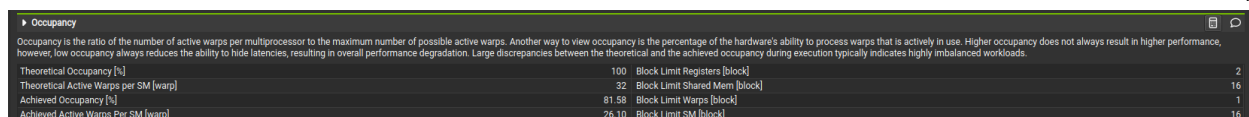
همانطور که مشاهده می شود، به میزان 115.06 GByte/second گذردهی حافظه استفاده شده، همینطور نشان داده میشود که L1 cache نیز به نسبت 0% hit شده که طبیعی هم هست، چرا که هیچ data reuse ای نداریم که این سطح cache بخواد hit شود.

اما در سطح L2 cache اوضاع متفاوت است و 56.17% hit rate داریم که این به خاطر محلیت مکانی دسترسی های نخ ها است.

به علت اینکه از shared memory در کرنل padding استفاده نکرده ایم، میزان خواندن و نوشتن در آن صفر است. خیر خطایی بنا بر دسترسی uncoalesced نمیگیریم.

چون از shared memory استفاده نکردیم، تعداد درخواست ها برای آن 0 است.

(د)



میزان theoretical occupancy برابر 100% و میزان achieved occupancy برابر 81.58% است.

همانطور که مشاهده میشود، میزان occupancy کسب شده برابر 81.91% است. از علت های آن میتوان به اینکه نخ ها محاسبه زیادی انجام نمیدهند و صرفا میخواهند یک خانه را از input بخوانند و در output بنویسند اشاره کرد، برای همین ممکن است زمان هایی باشد که هر SM برای تمام نخ هایی که بهش تخصیص داده شده منتظر داده از global memory است.

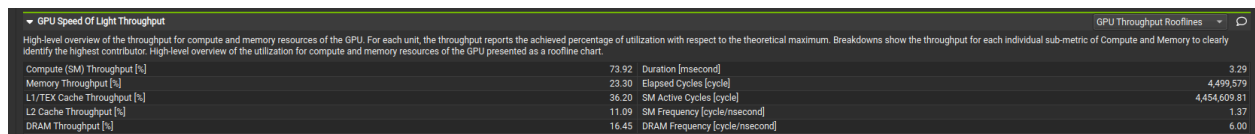
کرنل ۱:

کد: conv2D\_kernel

ابعاد ورودی: 3 \* 226 \* 226

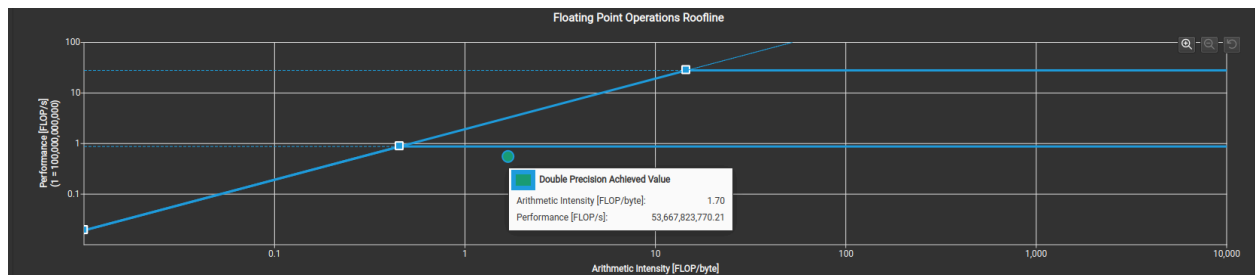
ابعاد خروجی: 64 \* 224 \* 224

(الف)



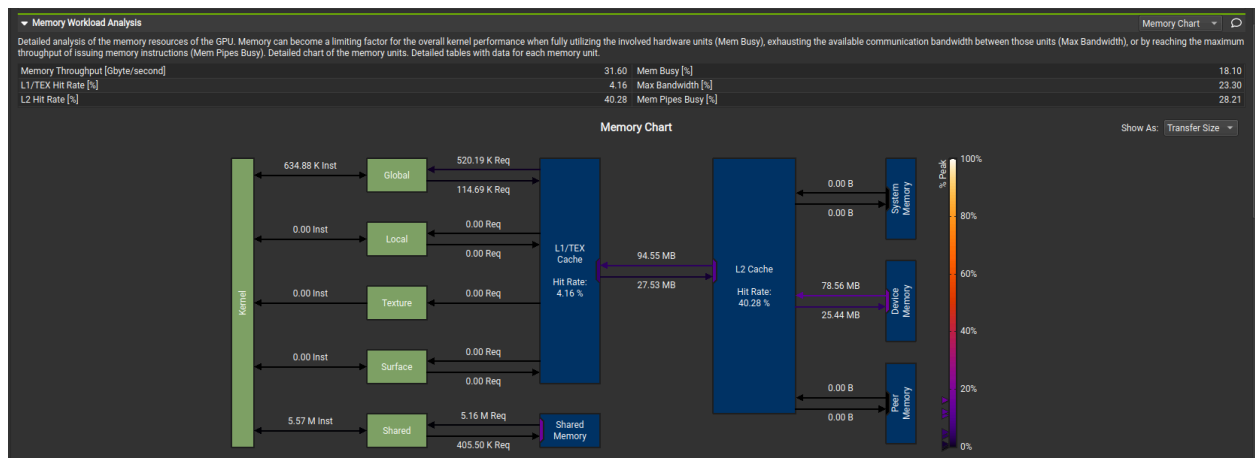
همانطور که مشاهده میشود در این کرنل، میزان compute throughput برابر 73.92% است، میزان memory throughput نیز برابر 23.30% است. همانطور که مشاهده میشود میزان throughput حافظه L1 cache برابر 36.20% است ولی برای L2 cache برابر 11.09% و برای DRAM برابر 16.45% است. این کرنل با توجه به گذردهی های گزارش شده در این tab، میتوان گفت که compute bound است.

(ب)



همانطور که مشاهده میشود، از آنجایی که از data type نوع double استفاده کرده ایم double precision achieved value به شدت بالا است و بسیار نزدیک به سقف double precision roofline هستیم که نشان دهنده این است که شدت حسابی بالایی داریم و compute bound هستیم، همانطور که tab قبلی نیز اشاره میکرد.

(ج)

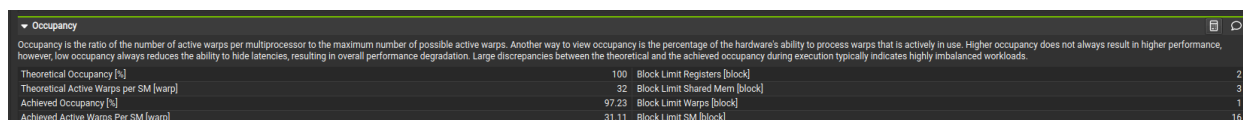


مشاهده میشود که میزان Memory throughput حدود 31.60 GByte/second است. همینطور برای L1 cache نیز برابر 4.16% است که آن هم پایین است، البته داده ای که نیاز است هر block استفاده کنند در shared memory ذخیره میشود و برای محاسبات استفاده میشود برای همین نباید انتظار استفاده بالا از L1 cache داشت چرا که نیاز به ذخیره داده هایی که دوباره به آن ها نیاز داریم در shared memory انجام شده. حافظه L2 cache میزان hit rate برابر 40.28% دارد. این عدد برای این بالاتر از L1 cache است که داده هایی که یک block نیاز داشته، توسط block دیگری نیز مورد نیاز بوده (مانند load margin ها، kernel ها) و به همین دلیل hit rate بالا تری کسب کرده ایم.

همانطور که مشاهده می شود، به میزان M 5.16 درخواست به shared memory برای خواندن داده زده میشود، پس استفاده از shared memory به صرفه بوده چرا که الگوریتم CNN به شدت data reuse دارد و با این کار جلوی هدر رفت زمان برای خواندن از global memory (یا از سلسله مراتب cache اگر خوش شانس باشیم) را گرفته ایم.

اینجا نیز خطاری برای uncoalesced memory access نداریم. همانطور که مشاهده میشود، تعداد درخواست ها برای خواندن از shared memory برابر M 5.16 درخواست است که در مقایسه برای خواندن از global memory که K 520.19 درخواست است به شدت بالاتر است.

(د)



Occupancy		
Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.		
Theoretical Occupancy [%]	100	Block Limit Registers [block]
Theoretical Active Warps per SM [warp]	32	Block Limit Shared Mem [block]
Achieved Occupancy [%]	97.23	Block Limit Warps [block]
Achieved Active Warps Per SM [warp]	31.11	Block Limit SM [block]

همانطور که مشاهده می شود، میزان occupancy برابر 97.23% است، که عدد بسیار مطلوبی است، و آن میزان حدود 3% نیز احتمالا به دلیل tail effect است که 100% نشده.

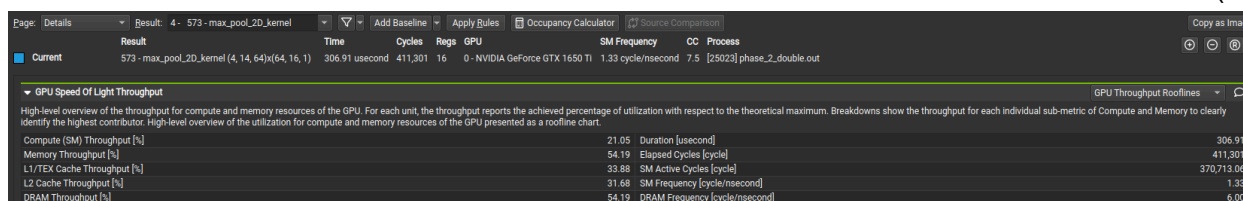
کرنل 4:

کد: max\_pool\_2D\_kernel

ابعاد ورودی: 224 \* 224 \* 64

ابعاد خروجی: 112 \* 112 \* 64

(الف)



GPU Speed of Light Throughput			
High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.			
Compute (SM) Throughput [%]	21.05	Duration [usecond]	306.91
Memory Throughput [%]	54.19	Elapsed Cycles [cycle]	411,301
L1/TEX Cache Throughput [%]	33.88	SM Active Cycles [cycle]	370,713.06
L2 Cache Throughput [%]	31.68	SM Frequency [cycle/nsecond]	1.33
DRAM Throughput [%]	54.19	DRAM Frequency [cycle/nsecond]	6.00

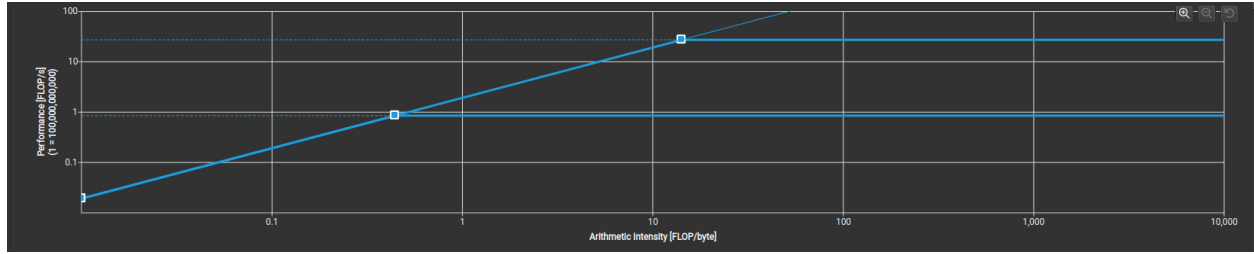
مانند padding، این kernel نیز محاسبات floating point خاصی ندارد و صرفا مقایسه است که بین 4 تا خانه مجاور در هر channel تصویر، کدام بیشترین مقدار را دارد.

مقدار compute throughput برابر 21.05% است و میزان Memory throughput نیز برابر 54.19% است.

میزان L1 cache throughput برابر 33.88% است و L2 cache throughput برابر 31.68%، برای DRAM نیز برابر 54.19% است.

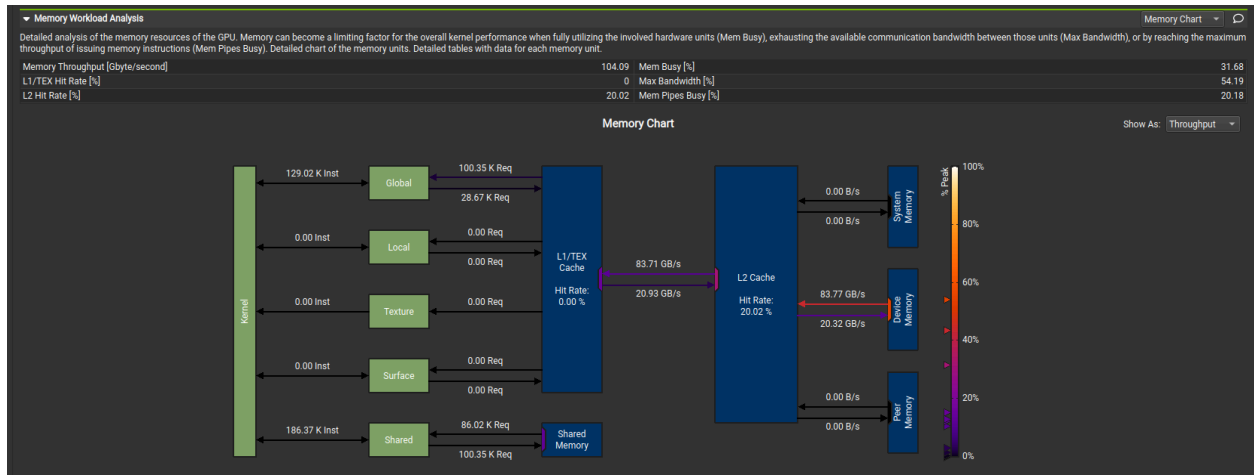
به علت اینکه این کرنل نیز صرفا کارش load کردن از memory، مقایسه و store کردن دوباره در memory است، میتوان گفت که memory bound است و کار محاسباتی ای انجام نمیدهد.

(ب)



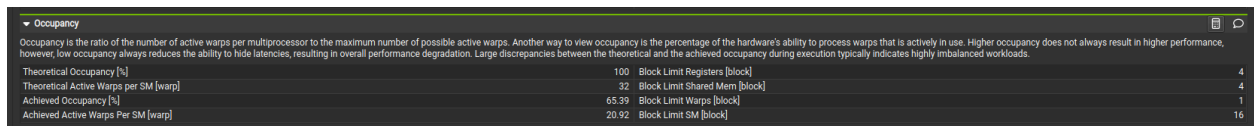
مانند padding، این کرنل نیز نقطه ای در roofline model برایش رسم نشده، چرا که شدت حسابی اش برابر 0 است.

(ج)



همانطور که مشاهده میشود، میزان memory throughput برابر 104.09 GByte/second است. میزان L1 cache hit rate برابر 0 است، که منطقی هم هست، چرا که هیچ data reuse ای نداریم که بخواهد L1 cache را hit کند. مقدار L2 cache hit rate برابر 20.02% است. همانطور که مشاهده میشود، میزان خواندن اطلاعات از حافظه device memory حدود 4 برابر میزان نوشتن در آن است، که به علت max pooling که از هر 4 خانه در تصویر ورودی یکی در خروجی نوشته میشود منطقی به نظر میرسد. همینطور مشاهده میشود که حدود K 100 درخواست برای خواندن از global memory داشتیم و حدود K 86 درخواست برای خواندن از shared memory، همینطور K 28.67 درخواست نوشتن در global memory و K 100.35 درخواست برای نوشتن در shared memory، پس میتوان گفت که data reuse به میزان خیلی زیادی بالا نیست که این به دلیل ذات خود max pooling است.

(د)



میزان theoretical occupancy در اینجا برابر 100% است اما به میزان 65.39% آن achieved است. یکی از علت های آن میتواند کاهش 4 برابری نخ ها (و در نتیجه warp ها) بعد از load کردن باشد که SM ها را بدون warp فعال باقی میگذارد. همینطور مشکل دیگر تقسیم بندی 2 بعدی تصاویر است (block ها ابعاد 32 در 32 دارند و نخ هایی در کناره ها بی کار باقی میمانند، چرا که ابعاد تصویر مضربی از 32 نیست)، و این کار باعث میشود که تعداد بیشتری از warp ها غیر فعال بمانند.

کرنل: 25

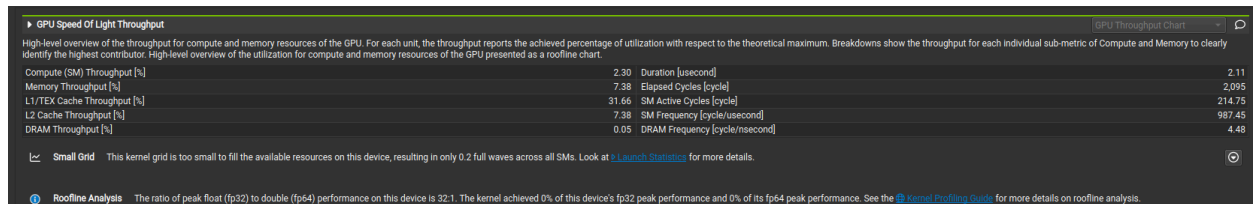
کد: zero\_all\_kernel

اندازه ورودی: 4096

اندازه خروجی: 4096 (همان بردار ورودی است اما همه درایه هایش 0 شده).

این کرنل یک کرنل کمکی برای پیاده سازی FC است، و تمام درایه های بردار خروجی را 0 میکند.

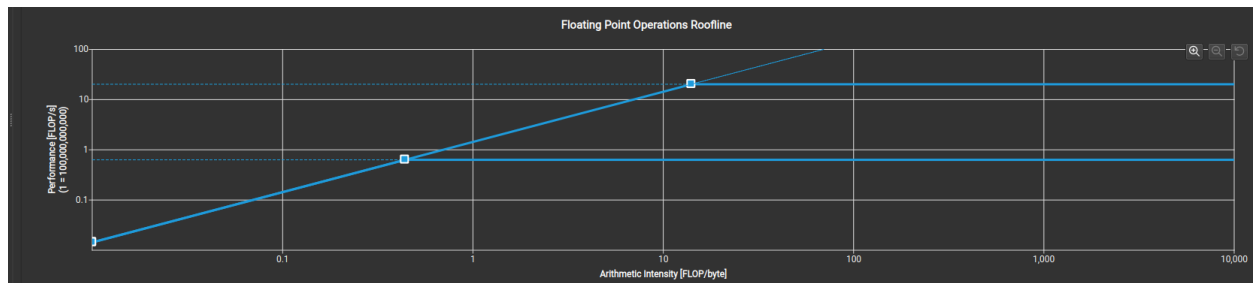
(الف)



مشاهده میشود که هم compute throughput برابر 2.30% است که به شدت پایین است، و هم Memory throughput برابر 7.38% و به شدت پایین، علت آن هم کوچک بودن بردار ورودی است و درواقع این کرنل از تمام ظرفیت gpu (نه از نظر محاسباتی و نه از نظر حافظه) استفاده نمیکند و این کار بهتر است سمت cpu هندل شود.

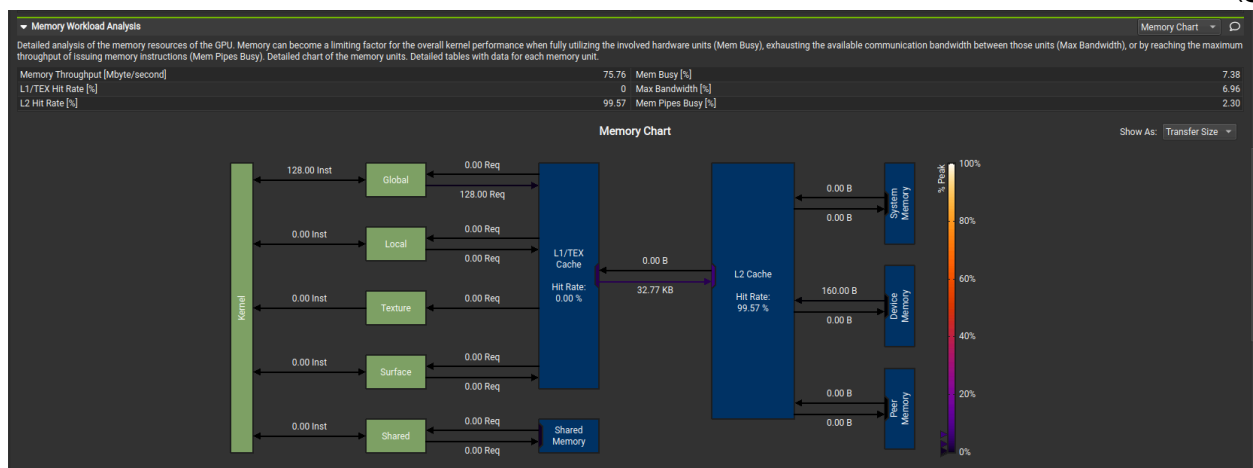
همینطور L1 cache throughput برابر 36.11%، L2 cache throughput برابر 7.38% و DRAM throughput برابر 0.05% است.

(ب)



همانطور که مشاهده میشود، چون این کرنل نیز کار محاسباتی انجام نمیدهد، در roofline model نقطه ای برای آن نشان داده نمیشود.

(ج)



همانطور که مشاهده میشود، Memory throughput برابر 75.76 GByte/second است که بسیار کم است، که این به دلیل کوچک بودن بردار نیز هست.

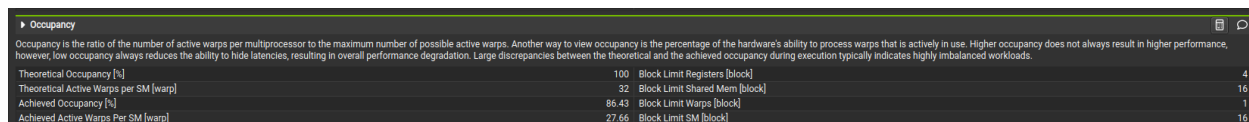
مقدار L1 cache hit rate برابر 0% است، چرا که هیچ data reuse ای نداریم و از قبل نیز این بردار در L1 cache نیوده (توسط block دیگری استفاده نشده).

مقدار L2 cache hit rate برابر 99.57 است.

درخواستی مبنی بر استفاده uncoalesced از حافظه نداریم.

از shared memory استفاده نشده و هیچ درخواستی هم به آن داده نمیشود.

(د)

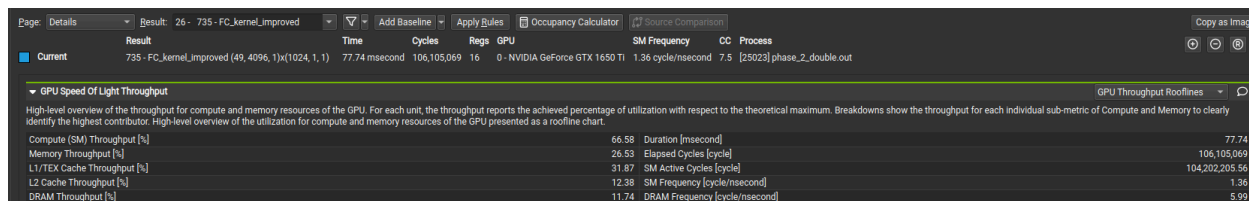


Occupancy			
Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.			
Theoretical Occupancy [%]	100	Block Limit Registers [block]	4
Theoretical Active Warps per SM [warp]	32	Block Limit Shared Mem [block]	16
Achieved Occupancy [%]	86.43	Block Limit Warps [block]	1
Achieved Active Warps Per SM [warp]	27.66	Block Limit SM [block]	16

همانطور که مشاهده میشود، میزان theoretical occupancy برابر 100% و میزان achieved occupancy برابر 86.43% است، علت اصلی آن را میتوان در کم بودن تعداد خانه های بردار دید و همینطور tail effect.

کرنل: 26

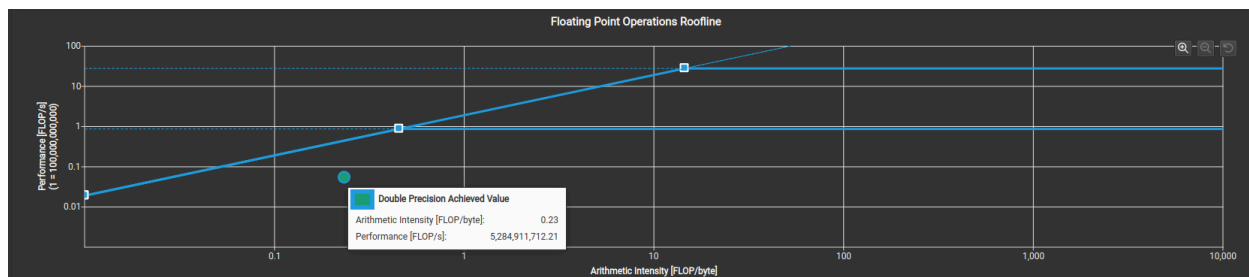
کد: FC\_kernel\_improved



GPU Speed Of Light Throughput			
High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.			
Compute (SM) Throughput [%]	66.58	Duration [msecond]	77.74
Memory Throughput [%]	26.53	Elapsed Cycles [cycle]	106,105,069
L1/TEX Cache Throughput [%]	31.87	SM Active Cycles [cycle]	104,202,205.56
L2 Cache Throughput [%]	12.38	SM Frequency [cycle/msecond]	1.36
DRAM Throughput [%]	11.74	DRAM Frequency [cycle/msecond]	5.99

همانطور که مشاهده میشود، compute throughput برابر 66% است و memory throughput برابر 26.53% است، با این اطلاعات میتوان نتیجه گرفت که compute bound نیستیم اما چندان هم از آن دور نیستیم، که به علت این است که از تمام ظرفیت محاسباتیمان استفاده نمیکنیم.

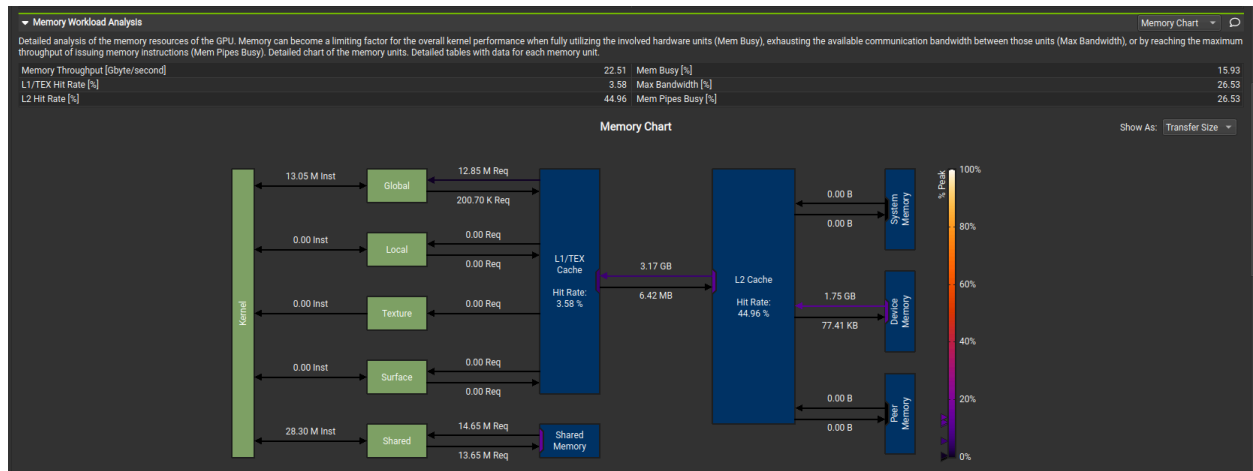
(ب)



همانطور که مشاهده میشود، شدت حسابی همچنان باید بیشتر شود تا compute bound شویم (الان در حدود 0.23 flop/byte است).

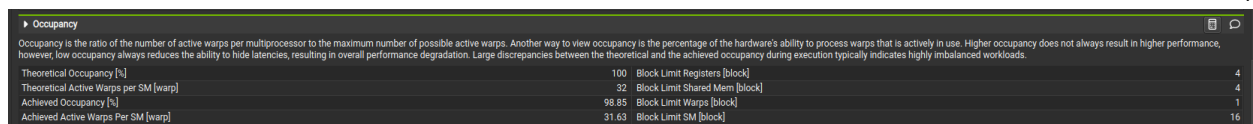
حدود  $10^9 * 5$  flop/second کارایی کرنل است.

(ج)



همانطور که مشاهده میشود، Memory throughput برابر 22.51 GByte/second است، که البته میزان پایینی است. میزان L1 cache hit rate برابر 3.58% و L2 cache hit rate برابر 44.96% است. همانطور که مشاهده میشود، به تعداد M 14.65 درخواست برای خواندن و M 13.65 درخواست برای نوشتن در shared memory داریم، در صورتی که 12.85 M درخواست برای خواندن و 200.70 K درخواست برای نوشتن در global memory داریم. علت آن هم این میتواند باشد که تعداد عملیات ها به شدت نزدیک به مرتبه تعداد خواندن اطلاعات از global memory است.

(د)



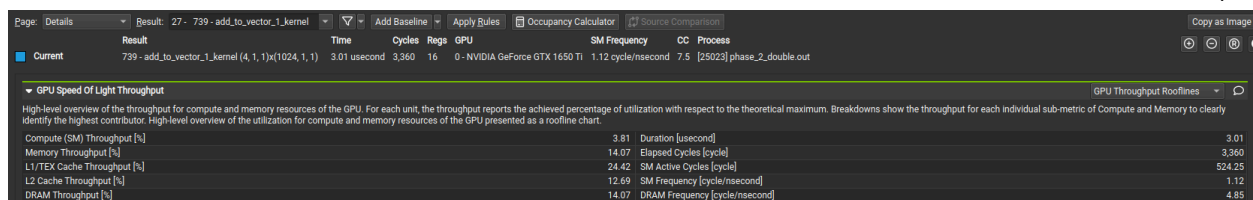
مشاهده میشود که theoretical occupancy برابر 100% ولی achieved occupancy برابر 98.85% است که به دلیل تعداد بالای نخ های استفاده شده است.

کرنل: 27

کد: add\_to\_vector\_1\_kernel

این کرنل برای جمع کردن درایه های خروجی FC\_kernel\_improved با bias های متناظر آن ها است و یک kernel کمکی است.

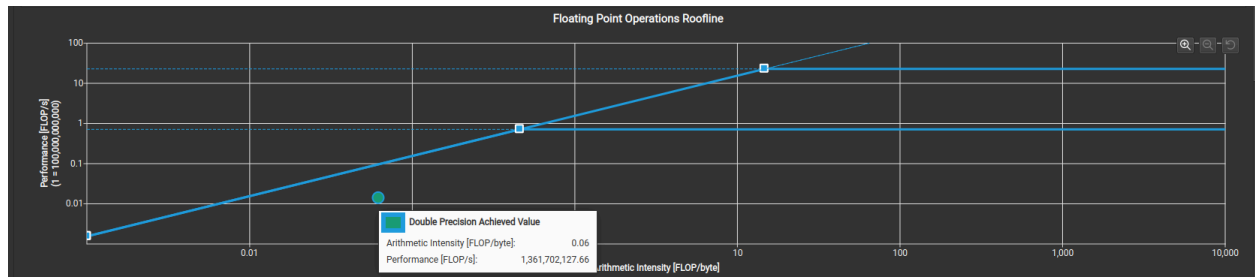
(الف)



همانطور که مشاهده میشود، compute throughput برابر 3.81% است و memory throughput برابر 14.07% است. یعنی از ظرفیت محاسباتی اصلا به میزان کافی استفاده نشده و این kernel نیز بهتر بوده سمت cpu پیاده سازی شود. میتوان با توجه به این دو عدد گفت که memory bound هست. میزان L1 cache throughput برابر 24.42%، میزان L2 cache throughput برابر 12.69% و میزان DRAM throughput برابر 14.07% است.

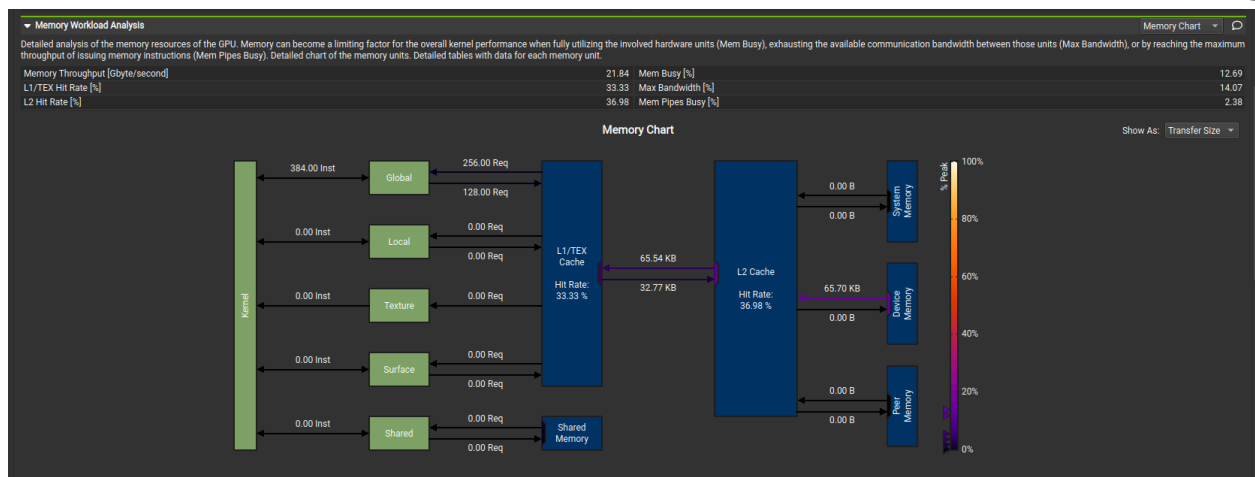
(ب)





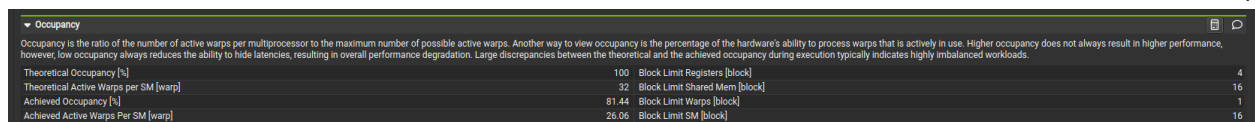
همانطور که پیش بینی شد، در محدوده memory bound قرار داریم و شدت حسابی ناچیز و حدود 0.06 است. کارایی کرنل برابر  $10^9 \times 1$  FLOP/second است.

(ج)



همانطور که مشاهده می شود، memory throughput برابر 21.84 GByte/second، میزان L1 cache hit rate برابر 33.33% و L2 cache hit rate برابر 36.98% است. از آنجایی که از shared memory استفاده نکرده ایم، درخواستی برای خواندن یا نوشتن به آن ارسال نشده و همه کارها با global memory بوده. اینجا نیز خطاری برای uncoalesced memory access نداریم.

(د)



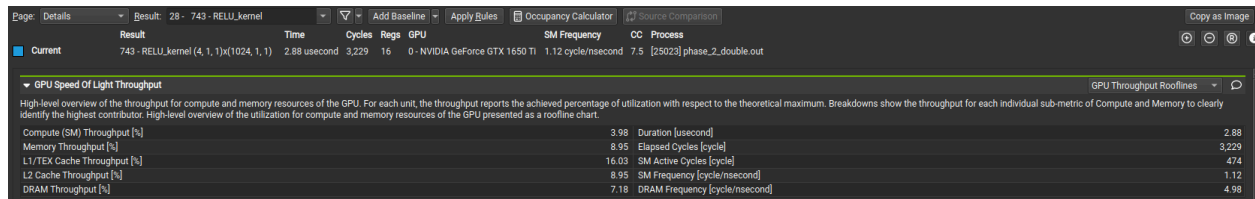
همانطور که مشاهده میشود، theoretical occupancy برابر 100% و achieved occupancy برابر 81.44% است که این میزان نشان دهنده تعداد پایین نخها (با آنکه به ازای هر درایه ورودی و خروجی یک نخ ساخته شده) است و بهتر بوده که این تابع سمت cpu پردازش شود.

کرنل: 28

کد: RELU\_kernel

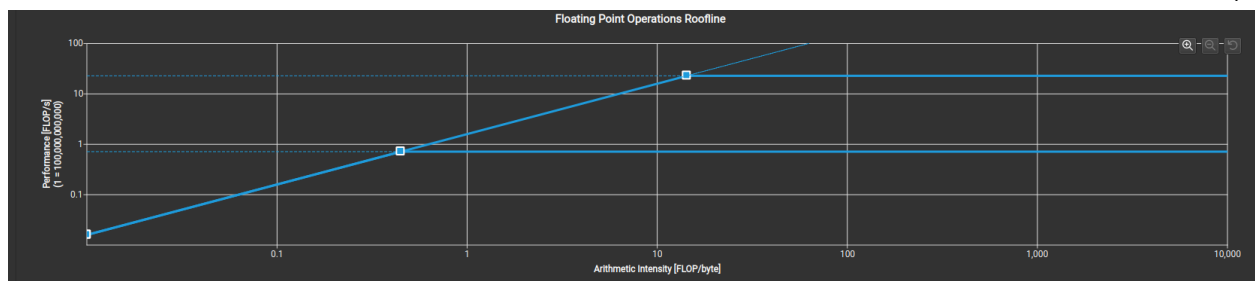
این کرنل نیز یک کرنل کمکی برای محاسبه FC است و هر درایه منفی را برابر 0 میکند.

(الف)



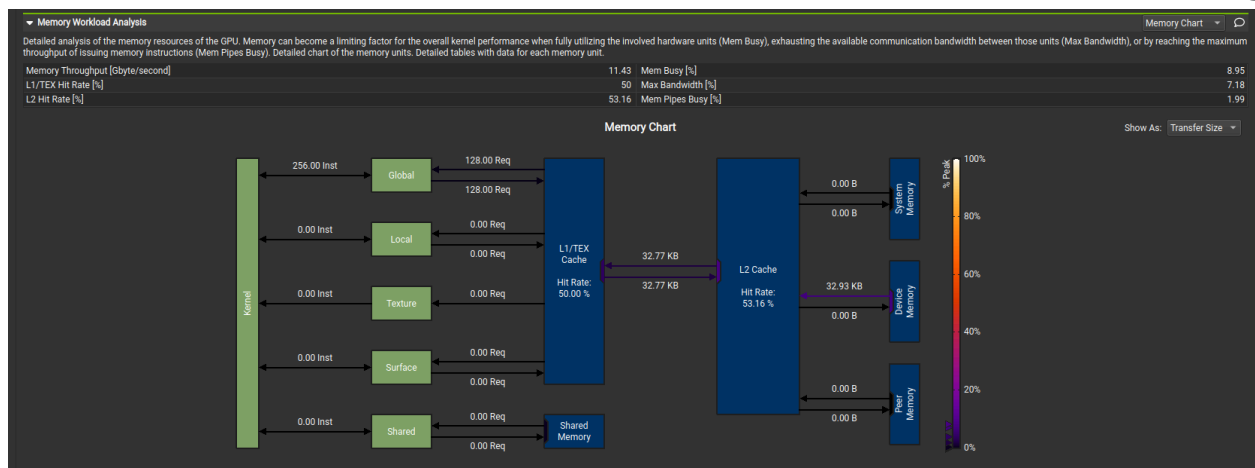
همانطور که مشاهده میشود، میزان compute throughput برابر 3.98% است، که به علت کوچک بودن اندازه ورودی و صرفا مقایسه با 0 به شدت پایین است. میزان memory throughput نیز برابر 8.95% است که نشان دهنده استفاده بسیار پایین از ظرفیت gpu است. این کرنل چون هیچ کار محاسباتی ای انجام نمیدهد، memory bound است.

(ب)



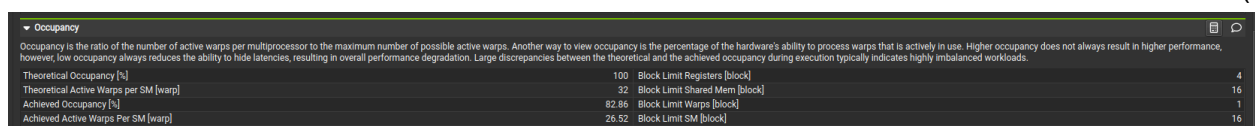
همانطور که مشاهده میشود، از آنجا که این کرنل کار محاسباتی ای ندارد در roofline model نیز نمایش داده نمیشود.

(ج)



همانطور که مشاهده میشود، memory throughput برابر 11.43 GByte/second است که به شدت پایین است، همینطور L1 cache hit rate برابر 50% و L2 cache hit rate برابر 53.16% است. از آنجا که از shared memory استفاده نشده، هیچ درخواستی برای خواندن یا نوشتن به آن ارسال نمیشود. اینجا نیز خطاری مبنی بر uncoalesced memory access نداریم.

(د)



همانطور که مشاهده میشود، theoretical occupancy برابر 100% اما achieved occupancy برابر 82.66% است که به علت کمبود تعداد نخ ها است و بهتر بوده این کار سمت cpu هندل شود ( یا در پیاده سازی FC ترکیب شود).